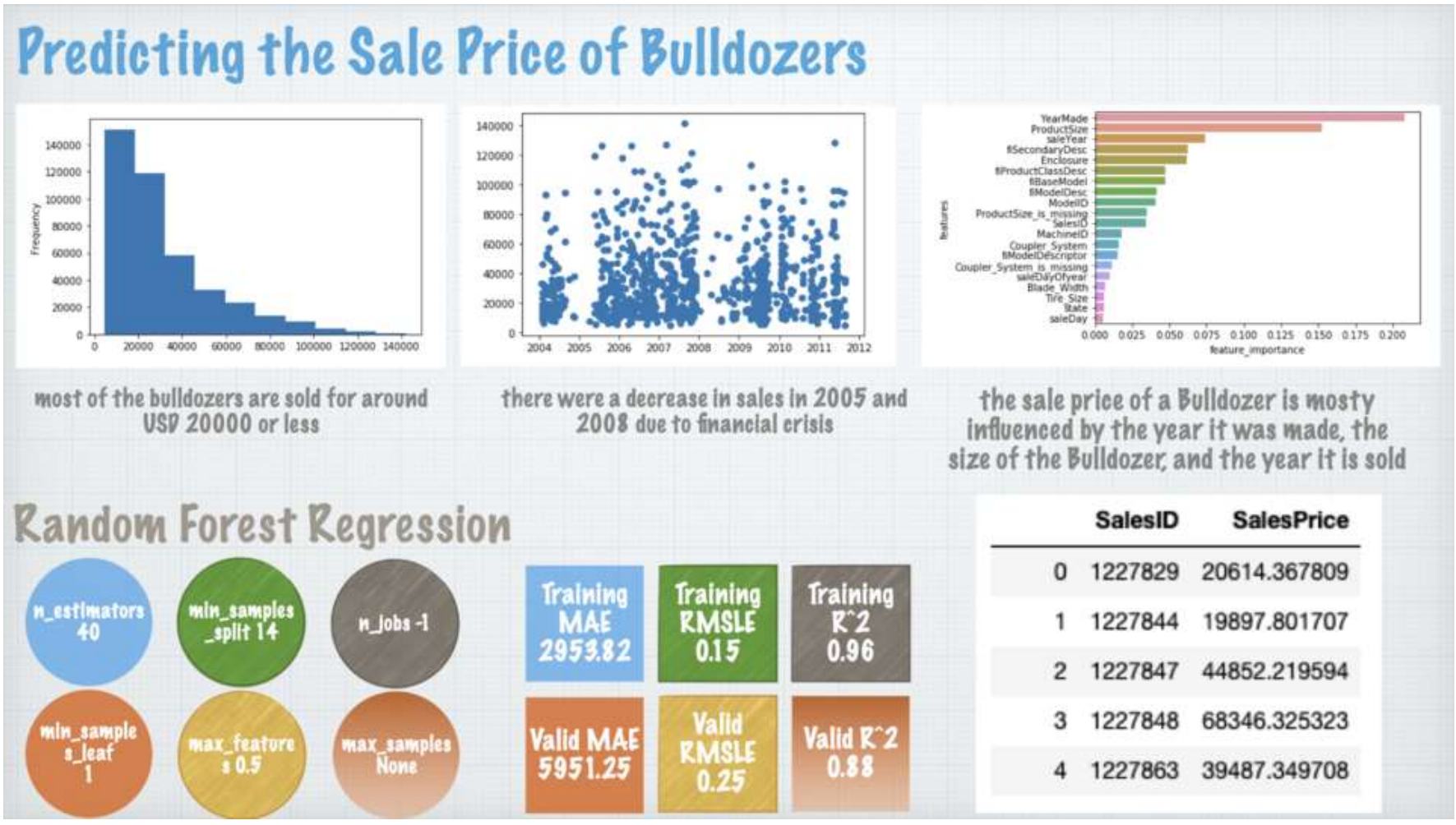
🚜 Predicting the sale price of Bulldozers using Machine Learning

Predict the auction sale price for a piece of heavy equipment to create a "blue book" for bulldozers. variable).?

1. Problem Definition

> How well can we predict the future sale price of a bulldozer, given its characteristic previous examples of how much similar bulldozers have been sold for?



	SalesID	SalesPrice
0	1227829	20614.367809
1	1227844	19897.801707
2	1227847	44852.219594
3	1227848	68346.325323
4	1227863	39487.349708

2. Data

Looking at the dataset from Kaggle, we can see it's a time series problem.

In this case, it's a historical sales data of bulldozers. Including details like model type, size, sale data and more.

There are 3 datasets:

1. Train.csv - Historical bulldozer sales examples up to 2011 (close to 400,000 examples with 50+ different attributes, including SalePrice which is the target variable).
2. Valid.csv - Historical bulldozer sales examples from January 1 2012 to April 30 2012 (close to 12,000 examples with the same attributes as Train.csv)
3. Test.csv - Historical bulldozer sales examples from May 1 2012 to November 2012 (close to 12,000 examples but without SalePrice column which is our target variable).

3. Evaluation

For this problem, Kaggle has set the evaluation metric to being root mean square log error (RMSLE). As with many regression evaluations, the goal will be to reduce this metric value as low as possible.

To see how well our model is doing, we'll calculate the RMSLE and then compare our results to others on the Kaggle leaderboard.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
```

In [2]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [3]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [4]:
```
# import train and validation set
df=pd.read_csv("bluebook-for-bulldozers/TrainAndValid.csv",low_memory=False)
```

In [5]:
```
df.head()
```

Out[5]:

	SalesID	SalePrice	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand	saledate	...
0	1139246	66000.0	999089	3157	121	3.0	2004	68.0	Low	11/16/2006 0:00	...
1	1139248	57000.0	117657	77	121	3.0	1996	4640.0	Low	3/26/2004 0:00	...
2	1139249	10000.0	434808	7009	121	3.0	2001	2838.0	High	2/26/2004 0:00	...
3	1139251	38500.0	1026470	332	121	3.0	2001	3486.0	High	5/19/2011 0:00	...
4	1139253	11000.0	1057373	17311	121	3.0	2007	722.0	Medium	7/23/2009 0:00	...

5 rows × 53 columns

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 412698 entries, 0 to 412697
Data columns (total 53 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   SalesID                   412698 non-null  int64
 1   SalePrice                 412698 non-null  float64
 2   MachineID                 412698 non-null  int64
 3   ModelID                   412698 non-null  int64
 4   datasource                412698 non-null  int64
 5   auctioneerID              392562 non-null  float64
 6   YearMade                  412698 non-null  int64
 7   MachineHoursCurrentMeter  147504 non-null  float64
 8   UsageBand                 73670 non-null   object
 9   saledate                  412698 non-null  object
 10  fiModelDesc               412698 non-null  object
 11  fiBaseModel               412698 non-null  object
 12  fiSecondaryDesc           271971 non-null  object
 13  fiModelSeries             58667 non-null   object
 14  fiModelDescriptor         74816 non-null   object
 15  ProductSize               196093 non-null  object
 16  fiProductClassDesc        412698 non-null  object
 17  state                     412698 non-null  object
 18  ProductGroup              412698 non-null  object
 19  ProductGroupDesc          412698 non-null  object
 20  Drive_System              107087 non-null  object
 21  Enclosure                 412364 non-null  object
 22  Forks                     197715 non-null  object
 23  Pad_Type                  81096 non-null   object
 24  Ride_Control              152728 non-null  object
 25  Stick                     81096 non-null   object
 26  Transmission              188007 non-null  object
 27  Turbocharged              81096 non-null   object
 28  Blade_Extension           25983 non-null   object
 29  Blade_Width               25983 non-null   object
 30  Enclosure_Type            25983 non-null   object
 31  Engine_Horsepower         25983 non-null   object
 32  Hydraulics                330133 non-null  object
 33  Pushblock                 25983 non-null   object
 34  Ripper                    106945 non-null  object
 35  Scarifier                 25994 non-null   object
 36  Tip_Control               25983 non-null   object
 37  Tire_Size                 97638 non-null   object
 38  Coupler                   220679 non-null  object
 39  Coupler_System            44974 non-null   object
 40  Grouser_Tracks            44875 non-null   object
 41  Hydraulics_Flow           44875 non-null   object
 42  Track_Type                102193 non-null  object
 43  Undercarriage_Pad_Width   102916 non-null  object
 44  Stick_Length              102261 non-null  object
 45  Thumb                     102332 non-null  object
 46  Pattern_Changer           102261 non-null  object
 47  Grouser_Type              102193 non-null  object
 48  Backhoe_Mounting          80712 non-null   object
 49  Blade_Type                81875 non-null   object
 50  Travel_Controls           81877 non-null   object
 51  Differential_Type         71564 non-null   object
 52  Steering_Controls         71522 non-null   object
dtypes: float64(3), int64(5), object(45)
memory usage: 166.9+ MB
```

In [7]:
```
df.isna().sum()
```

Out[7]:
```
SalesID                     0
SalePrice                   0
MachineID                   0
ModelID                     0
datasource                  0
auctioneerID            20136
YearMade                    0
MachineHoursCurrentMeter  265194
UsageBand                 339028
saledate                    0
fiModelDesc                 0
fiBaseModel                 0
fiSecondaryDesc           140727
fiModelSeries             354031
fiModelDescriptor         337882
ProductSize               216605
fiProductClassDesc          0
state                       0
ProductGroup                0
ProductGroupDesc            0
Drive_System              305611
Enclosure                    334
Forks                     214983
Pad_Type                  331602
Ride_Control              259970
Stick                     331602
Transmission              224691
Turbocharged              331602
Blade_Extension           386715
Blade_Width               386715
Enclosure_Type            386715
Engine_Horsepower         386715
Hydraulics                 82565
Pushblock                 386715
Ripper                    305753
Scarifier                 386704
Tip_Control               386715
Tire_Size                 315060
Coupler                   192019
Coupler_System            367724
Grouser_Tracks            367823
Hydraulics_Flow           367823
Track_Type                310505
Undercarriage_Pad_Width   309782
Stick_Length              310437
Thumb                     310366
Pattern_Changer           310437
Grouser_Type              310505
Backhoe_Mounting          331986
Blade_Type                330823
Travel_Controls           330821
Differential_Type         341134
Steering_Controls         341176
dtype: int64
```

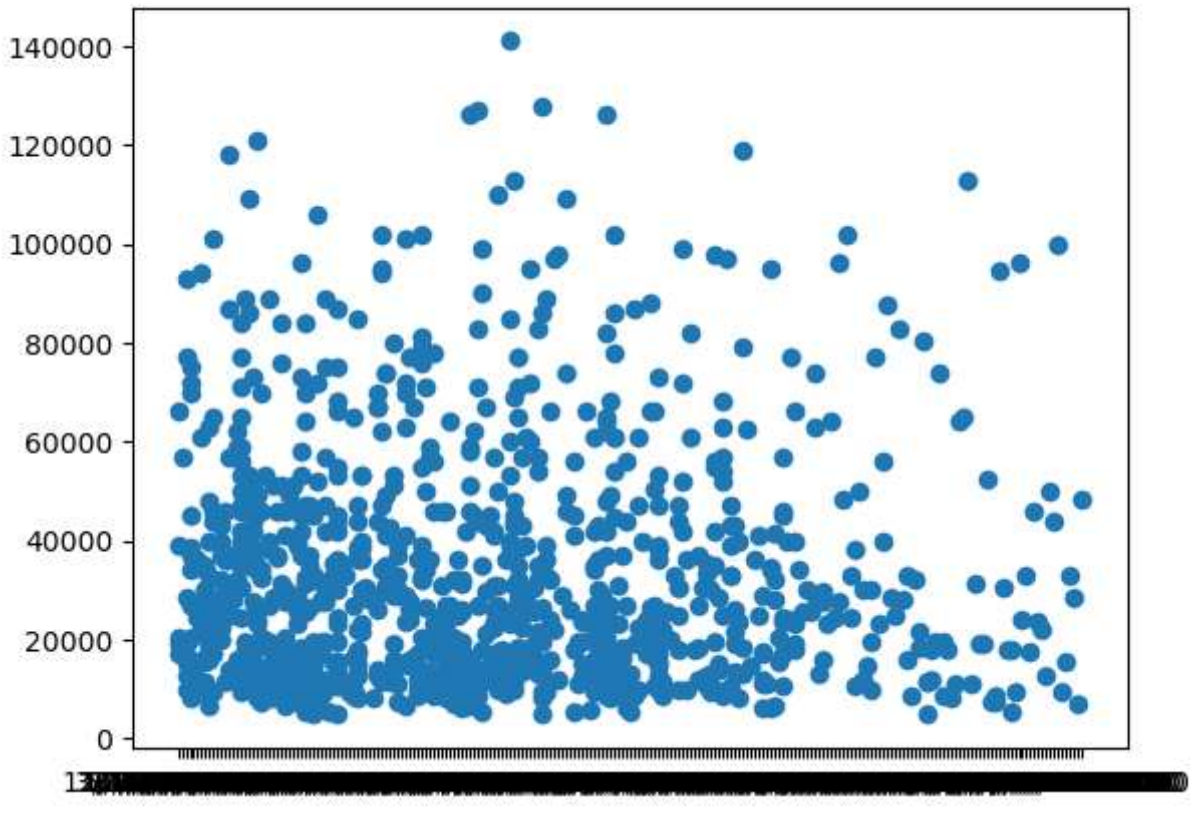
df.columns()

In [8]:
```
df.describe()
```

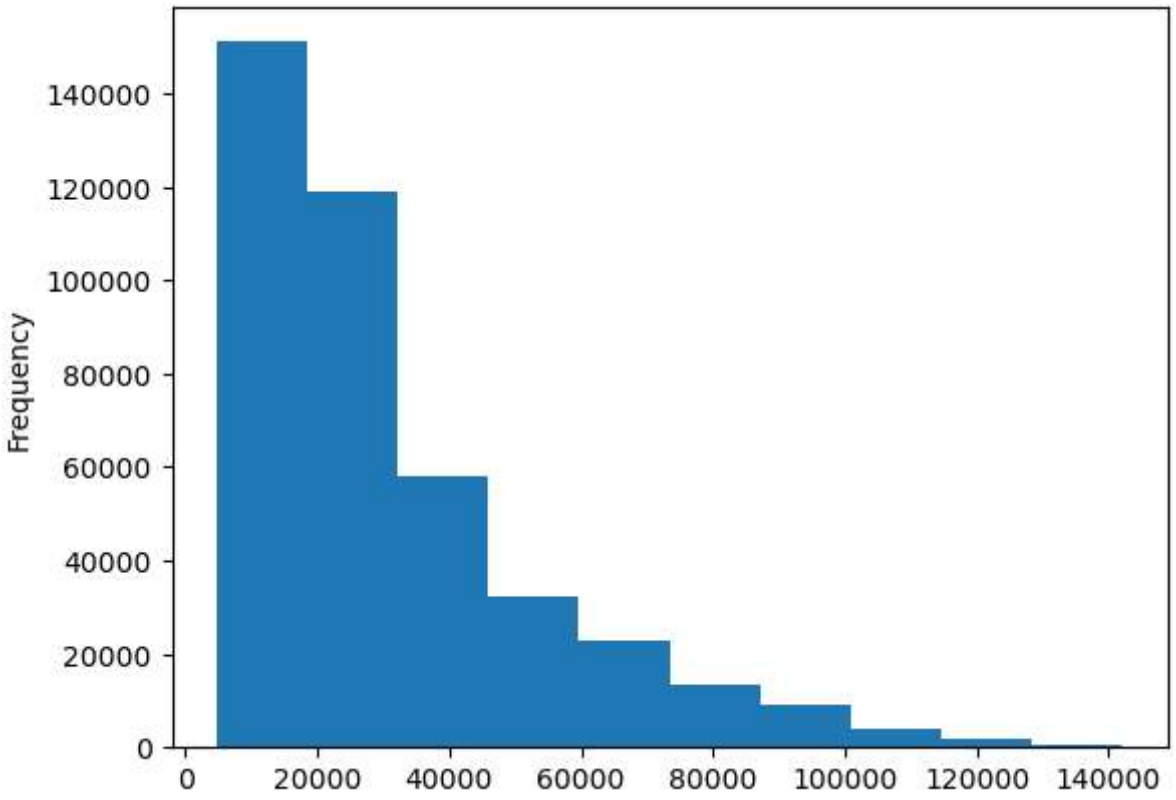
Out[8]:

	SalesID	SalePrice	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMet
count	4.126980e+05	412698.000000	4.126980e+05	412698.000000	412698.000000	392562.000000	412698.000000	1.475040e+(
mean	2.011161e+06	31215.181414	1.230061e+06	6947.201828	135.169361	6.585268	1899.049637	3.522988e+(
std	1.080068e+06	23141.743695	4.539533e+05	6280.824982	9.646749	17.158409	292.190243	2.716993e+(
min	1.139246e+06	4750.000000	0.000000e+00	28.000000	121.000000	0.000000	1000.000000	0.000000e+(
25%	1.421898e+06	14500.000000	1.088593e+06	3261.000000	132.000000	1.000000	1985.000000	0.000000e+(
50%	1.645852e+06	24000.000000	1.284397e+06	4605.000000	132.000000	2.000000	1995.000000	0.000000e+(
75%	2.261012e+06	40000.000000	1.478079e+06	8899.000000	136.000000	4.000000	2001.000000	3.209000e+(
max	6.333349e+06	142000.000000	2.486330e+06	37198.000000	173.000000	99.000000	2014.000000	2.483300e+(

In [9]:
```
fig, ax=plt.subplots()
ax.scatter(df["saledate"][:1000],df["SalePrice"][:1000]);
```



In [10]:
```
df["SalePrice"].plot.hist();
```



In [11]:
```
df.saledate[:100]
```

Out[11]:
```
0     11/16/2006 0:00
1      3/26/2004 0:00
2      2/26/2004 0:00
3      5/19/2011 0:00
4      7/23/2009 0:00
           ...
95    12/15/2005 0:00
96     1/29/2004 0:00
97     9/18/2008 0:00
98     11/3/2005 0:00
99      6/1/2006 0:00
Name: saledate, Length: 100, dtype: object
```

Parsing Dates

When we work with time serves data, we want to enrich the time and date component as much as possible.

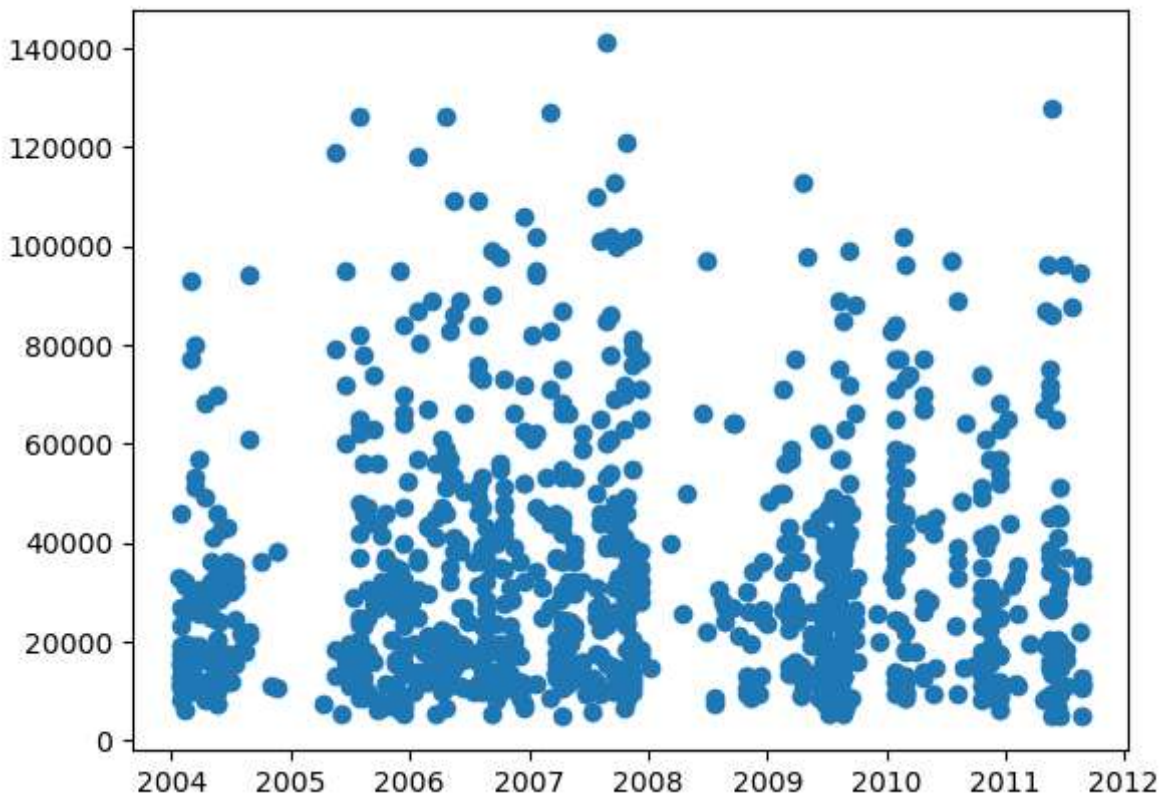
We can do that by telling pandas which of our columns has dates in it using the `parse_date` parameter

In [12]:
```
#import the data again but this time parse dates
df=pd.read_csv("bluebook-for-bulldozers/TrainAndValid.csv",low_memory=False,parse_dates=["saledate"])
```

In [13]:
```
df.saledate[:100]
```

Out[13]:
```
0    2006-11-16
1    2004-03-26
2    2004-02-26
3    2011-05-19
4    2009-07-23
        ...
95   2005-12-15
96   2004-01-29
97   2008-09-18
98   2005-11-03
99   2006-06-01
Name: saledate, Length: 100, dtype: datetime64[ns]
```

In [14]:
```
fig, ax=plt.subplots()
ax.scatter(df["saledate"][:1000],df["SalePrice"][:1000]);
```



Sort DataFrame by saledate

When working with time series data, it's good idea to sort it by date.

In [15]:
```
#SOrt DataFrame in data order
df.sort_values(by=["saledate"],inplace=True,ascending=True)
df.saledate.head()
```

Out[15]:
```
205615   1989-01-17
274835   1989-01-31
141296   1989-01-31
212552   1989-01-31
62755    1989-01-31
Name: saledate, dtype: datetime64[ns]
```

Make a copy of original dataframe

We make a copy of the original Dataframe so when we manipulate the copy, we've still got our original data.

In [16]:
```
# Make a copy of original dataframe to perform edits on
df_temp=df.copy()
df_temp.head()
```

Out[16]:

	SalesID	SalePrice	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand	saledate
205615	1646770	9500.0	1126363	8434	132	18.0	1974	NaN	NaN	1989-01-17
274835	1821514	14000.0	1194089	10150	132	99.0	1980	NaN	NaN	1989-01-31
141296	1505138	50000.0	1473654	4139	132	99.0	1978	NaN	NaN	1989-01-31
212552	1671174	16000.0	1327630	8591	132	99.0	1980	NaN	NaN	1989-01-31
62755	1329056	22000.0	1336053	4089	132	99.0	1984	NaN	NaN	1989-01-31

5 rows × 53 columns

Add a parameter for `saledown` column

In [17]:
```
df_temp[:1].saledate.dt.year
```

Out[17]:
```
205615    1989
Name: saledate, dtype: int32
```

In [18]:
```
df_temp["saleYear"]=df_temp.saledate.dt.year
df_temp["saleMonth"]=df_temp.saledate.dt.month
df_temp["saleDay"]=df_temp.saledate.dt.day
df_temp["saleDayofWeek"]=df_temp.saledate.dt.dayofweek
df_temp["saleDayofYear"]=df_temp.saledate.dt.dayofyear
```

In [19]:
```
df_temp.head()
```

Out[19]:

	SalesID	SalePrice	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand	saledate
205615	1646770	9500.0	1126363	8434	132	18.0	1974	NaN	NaN	1989-01-17
274835	1821514	14000.0	1194089	10150	132	99.0	1980	NaN	NaN	1989-01-31
141296	1505138	50000.0	1473654	4139	132	99.0	1978	NaN	NaN	1989-01-31
212552	1671174	16000.0	1327630	8591	132	99.0	1980	NaN	NaN	1989-01-31
62755	1329056	22000.0	1336053	4089	132	99.0	1984	NaN	NaN	1989-01-31

5 rows × 58 columns

In [20]:
```
# Now we have enriched to our dataframe with date and time feature, so we can remove saledate column from our duplicate data
df_temp.drop("saledate",axis=1,inplace=True)
```

In [21]:
```
#check the value of different column
df_temp.state.value_counts()
```

Out[21]:
```
state
Florida           67320
Texas             53110
California        29761
Washington        16222
Georgia           14633
Maryland          13322
Mississippi       13240
Ohio              12369
Illinois          11540
Colorado          11529
New Jersey        11156
North Carolina    10636
Tennessee         10298
Alabama           10292
Pennsylvania      10234
South Carolina     9951
Arizona            9364
New York           8639
Connecticut        8276
Minnesota          7885
Missouri           7178
Nevada             6932
Louisiana          6627
Kentucky           5351
Maine              5096
Indiana            4124
Arkansas           3933
New Mexico         3631
Utah               3046
Unspecified        2801
Wisconsin          2745
New Hampshire      2738
Virginia           2353
Idaho              2025
Oregon             1911
Michigan           1831
Wyoming            1672
Montana            1336
Iowa               1336
Oklahoma           1326
Nebraska            866
West Virginia       840
Kansas              667
Delaware            510
North Dakota        480
Alaska              430
Massachusetts       347
Vermont             300
South Dakota        244
Hawaii              118
Rhode Island         83
Puerto Rico          42
Washington DC         2
Name: count, dtype: int64
```

In [22]:
```
df_temp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 412698 entries, 205615 to 409203
Data columns (total 57 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   SalesID                   412698 non-null  int64
 1   SalePrice                 412698 non-null  float64
 2   MachineID                 412698 non-null  int64
 3   ModelID                   412698 non-null  int64
 4   datasource                412698 non-null  int64
 5   auctioneerID              392562 non-null  float64
 6   YearMade                  412698 non-null  int64
 7   MachineHoursCurrentMeter  147504 non-null  float64
 8   UsageBand                 73670 non-null   object
 9   fiModelDesc               412698 non-null  object
 10  fiBaseModel               412698 non-null  object
 11  fiSecondaryDesc           271971 non-null  object
 12  fiModelSeries             58667 non-null   object
 13  fiModelDescriptor         74816 non-null   object
 14  ProductSize               196093 non-null  object
 15  fiProductClassDesc        412698 non-null  object
 16  state                     412698 non-null  object
 17  ProductGroup              412698 non-null  object
 18  ProductGroupDesc          412698 non-null  object
 19  Drive_System              107087 non-null  object
 20  Enclosure                 412364 non-null  object
 21  Forks                     197715 non-null  object
 22  Pad_Type                  81096 non-null   object
 23  Ride_Control              152728 non-null  object
 24  Stick                     81096 non-null   object
 25  Transmission              188007 non-null  object
 26  Turbocharged              81096 non-null   object
 27  Blade_Extension           25983 non-null   object
 28  Blade_Width               25983 non-null   object
 29  Enclosure_Type            25983 non-null   object
 30  Engine_Horsepower         25983 non-null   object
 31  Hydraulics                330133 non-null  object
 32  Pushblock                 25983 non-null   object
 33  Ripper                    106945 non-null  object
 34  Scarifier                 25994 non-null   object
 35  Tip_Control               25983 non-null   object
 36  Tire_Size                 97638 non-null   object
 37  Coupler                   220679 non-null  object
 38  Coupler_System            44974 non-null   object
 39  Grouser_Tracks            44875 non-null   object
 40  Hydraulics_Flow           44875 non-null   object
 41  Track_Type                102193 non-null  object
 42  Undercarriage_Pad_Width   102916 non-null  object
 43  Stick_Length              102261 non-null  object
 44  Thumb                     102332 non-null  object
 45  Pattern_Changer           102261 non-null  object
 46  Grouser_Type              102193 non-null  object
 47  Backhoe_Mounting          80712 non-null   object
 48  Blade_Type                81875 non-null   object
 49  Travel_Controls           81877 non-null   object
 50  Differential_Type         71564 non-null   object
 51  Steering_Controls         71522 non-null   object
 52  saleYear                  412698 non-null  int32
 53  saleMonth                 412698 non-null  int32
 54  saleDay                   412698 non-null  int32
 55  saleDayofWeek             412698 non-null  int32
 56  saleDayofYear             412698 non-null  int32
dtypes: float64(3), int32(5), int64(5), object(44)
memory usage: 174.7+ MB
```

5.Modeling

We have done EDA (we could always do more) but let's do some model-driven EDA.

Converting Stirng to category integer

One way to turn all the data into numbers is by pandas categories.

In [23]:
```
pd.api.types.is_string_dtype(df_temp["UsageBand"].dtype)
```

Out[23]:
```
True
```

In [24]:
```
#find column which contains string
for label, content in df_temp.items():
    if pd.api.types.is_string_dtype(content.dtype):
        print(label)
```

```
UsageBand
fiModelDesc
fiBaseModel
fiSecondaryDesc
fiModelSeries
fiModelDescriptor
ProductSize
fiProductClassDesc
state
ProductGroup
ProductGroupDesc
Drive_System
Enclosure
Forks
Pad_Type
Ride_Control
Stick
Transmission
Turbocharged
Blade_Extension
Blade_Width
Enclosure_Type
Engine_Horsepower
Hydraulics
Pushblock
Ripper
Scarifier
Tip_Control
Tire_Size
Coupler
Coupler_System
Grouser_Tracks
Hydraulics_Flow
Track_Type
Undercarriage_Pad_Width
Stick_Length
Thumb
Pattern_Changer
Grouser_Type
Backhoe_Mounting
Blade_Type
Travel_Controls
Differential_Type
Steering_Controls
```

In [25]:
```
#converting all string column to category
for label, content in df_temp.items():
    if pd.api.types.is_string_dtype(content.dtype):
        df_temp[label]=content.astype("category").cat.as_ordered()
```

In [26]:
```
df_temp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 412698 entries, 205615 to 409203
Data columns (total 57 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   SalesID                   412698 non-null  int64
 1   SalePrice                 412698 non-null  float64
 2   MachineID                 412698 non-null  int64
 3   ModelID                   412698 non-null  int64
 4   datasource                412698 non-null  int64
 5   auctioneerID              392562 non-null  float64
 6   YearMade                  412698 non-null  int64
 7   MachineHoursCurrentMeter  147504 non-null  float64
 8   UsageBand                 73670 non-null   category
 9   fiModelDesc               412698 non-null  category
 10  fiBaseModel               412698 non-null  category
 11  fiSecondaryDesc           271971 non-null  category
 12  fiModelSeries             58667 non-null   category
 13  fiModelDescriptor         74816 non-null   category
 14  ProductSize               196093 non-null  category
 15  fiProductClassDesc        412698 non-null  category
 16  state                     412698 non-null  category
 17  ProductGroup              412698 non-null  category
 18  ProductGroupDesc          412698 non-null  category
 19  Drive_System              107087 non-null  category
 20  Enclosure                 412364 non-null  category
 21  Forks                     197715 non-null  category
 22  Pad_Type                  81096 non-null   category
 23  Ride_Control              152728 non-null  category
 24  Stick                     81096 non-null   category
 25  Transmission              188007 non-null  category
 26  Turbocharged              81096 non-null   category
 27  Blade_Extension           25983 non-null   category
 28  Blade_Width               25983 non-null   category
 29  Enclosure_Type            25983 non-null   category
 30  Engine_Horsepower         25983 non-null   category
 31  Hydraulics                330133 non-null  category
 32  Pushblock                 25983 non-null   category
 33  Ripper                    106945 non-null  category
 34  Scarifier                 25994 non-null   category
 35  Tip_Control               25983 non-null   category
 36  Tire_Size                 97638 non-null   category
 37  Coupler                   220679 non-null  category
 38  Coupler_System            44974 non-null   category
 39  Grouser_Tracks            44875 non-null   category
 40  Hydraulics_Flow           44875 non-null   category
 41  Track_Type                102193 non-null  category
 42  Undercarriage_Pad_Width   102916 non-null  category
 43  Stick_Length              102261 non-null  category
 44  Thumb                     102332 non-null  category
 45  Pattern_Changer           102261 non-null  category
 46  Grouser_Type              102193 non-null  category
 47  Backhoe_Mounting          80712 non-null   category
 48  Blade_Type                81875 non-null   category
 49  Travel_Controls           81877 non-null   category
 50  Differential_Type         71564 non-null   category
 51  Steering_Controls         71522 non-null   category
 52  saleYear                  412698 non-null  int32
 53  saleMonth                 412698 non-null  int32
 54  saleDay                   412698 non-null  int32
 55  saleDayofWeek             412698 non-null  int32
 56  saleDayofYear             412698 non-null  int32
dtypes: category(44), float64(3), int32(5), int64(5)
memory usage: 55.4 MB
```

Thanks to pandas Categories we now have away to access all our data in the form of numbers.

But we still have a bunch of missing values.

In [27]:
```
#Check missing data
df_temp.isna().sum()/len(df_temp)
```

Out[27]:
```
SalesID                   0.000000
SalePrice                 0.000000
MachineID                 0.000000
ModelID                   0.000000
datasource                0.000000
auctioneerID              0.048791
YearMade                  0.000000
MachineHoursCurrentMeter  0.642586
UsageBand                 0.821492
fiModelDesc               0.000000
fiBaseModel               0.000000
fiSecondaryDesc           0.340993
fiModelSeries             0.857845
fiModelDescriptor         0.818715
ProductSize               0.524851
fiProductClassDesc        0.000000
state                     0.000000
ProductGroup              0.000000
ProductGroupDesc          0.000000
Drive_System              0.740520
Enclosure                 0.000809
Forks                     0.520921
Pad_Type                  0.803498
Ride_Control              0.629928
Stick                     0.803498
Transmission              0.544444
Turbocharged              0.803498
Blade_Extension           0.937041
Blade_Width               0.937041
Enclosure_Type            0.937041
Engine_Horsepower         0.937041
Hydraulics                0.200062
Pushblock                 0.937041
Ripper                    0.740864
Scarifier                 0.937014
Tip_Control               0.937041
Tire_Size                 0.763415
Coupler                   0.465277
Coupler_System            0.891024
Grouser_Tracks            0.891264
Hydraulics_Flow           0.891264
Track_Type                0.752378
Undercarriage_Pad_Width   0.750626
Stick_Length              0.752213
Thumb                     0.752041
Pattern_Changer           0.752213
Grouser_Type              0.752378
Backhoe_Mounting          0.804428
Blade_Type                0.801610
Travel_Controls           0.801606
Differential_Type         0.826595
Steering_Controls         0.826697
saleYear                  0.000000
saleMonth                 0.000000
saleDay                   0.000000
saleDayofWeek             0.000000
saleDayofYear             0.000000
dtype: float64
```

In [28]:
```
df_temp.state.cat.categories
```

Out[28]:
```
Index(['Alabama', 'Alaska', 'Arizona', 'Arkansas', 'California', 'Colorado',
       'Connecticut', 'Delaware', 'Florida', 'Georgia', 'Hawaii', 'Idaho',
       'Illinois', 'Indiana', 'Iowa', 'Kansas', 'Kentucky', 'Louisiana',
       'Maine', 'Maryland', 'Massachusetts', 'Michigan', 'Minnesota',
       'Mississippi', 'Missouri', 'Montana', 'Nebraska', 'Nevada',
       'New Hampshire', 'New Jersey', 'New Mexico', 'New York',
       'North Carolina', 'North Dakota', 'Ohio', 'Oklahoma', 'Oregon',
       'Pennsylvania', 'Puerto Rico', 'Rhode Island', 'South Carolina',
       'South Dakota', 'Tennessee', 'Texas', 'Unspecified', 'Utah', 'Vermont',
       'Virginia', 'Washington', 'Washington DC', 'West Virginia', 'Wisconsin',
       'Wyoming'],
      dtype='object')
```

In [29]:
```
df_temp.state.value_counts()
```

Out[29]:
```
state
Florida           67320
Texas             53110
California        29761
Washington        16222
Georgia           14633
Maryland          13322
Mississippi       13240
Ohio              12369
Illinois          11540
Colorado          11529
New Jersey        11156
North Carolina    10636
Tennessee         10298
Alabama           10292
Pennsylvania      10234
South Carolina     9951
Arizona            9364
New York           8639
Connecticut        8276
Minnesota          7885
Missouri           7178
Nevada             6932
Louisiana          6627
Kentucky           5351
Maine              5096
Indiana            4124
Arkansas           3933
New Mexico         3631
Utah               3046
Unspecified        2801
Wisconsin          2745
New Hampshire      2738
Virginia           2353
Idaho              2025
Oregon             1911
Michigan           1831
Wyoming            1672
Montana            1336
Iowa               1336
Oklahoma           1326
Nebraska            866
West Virginia       840
Kansas              667
Delaware            510
North Dakota        480
Alaska              430
Massachusetts       347
Vermont             300
South Dakota        244
Hawaii              118
Rhode Island         83
Puerto Rico          42
Washington DC         2
Name: count, dtype: int64
```

In [30]:
```
df_temp.state.cat.codes
```

Out[30]:
```
205615    43
274835     8
141296     8
212552     8
62755      8
          ..
410879     4
412476     4
411927     4
407124     4
409203     4
Length: 412698, dtype: int8
```

Save preprocessed Data

In [31]:
```
#Export current temp Dataframe
df_temp.to_csv("bluebook-for-bulldozers/train_temp.csv",index=False)
```

In [32]:
```
#import preprocessed data
df_temp=pd.read_csv("bluebook-for-bulldozers/train_temp.csv",low_memory=False)
```

In [33]:
```
df_temp.isna().sum()
```

Out[33]:
```
SalesID                       0
SalePrice                     0
MachineID                     0
ModelID                       0
datasource                    0
auctioneerID              20136
YearMade                      0
MachineHoursCurrentMeter  265194
UsageBand                 339028
fiModelDesc                   0
fiBaseModel                   0
fiSecondaryDesc           140727
fiModelSeries             354031
fiModelDescriptor         337882
ProductSize               216605
fiProductClassDesc            0
state                         0
ProductGroup                  0
ProductGroupDesc              0
Drive_System              305611
Enclosure                    334
Forks                     214983
Pad_Type                  331602
Ride_Control              259970
Stick                     331602
Transmission              224691
Turbocharged              331602
Blade_Extension           386715
Blade_Width               386715
Enclosure_Type            386715
Engine_Horsepower         386715
Hydraulics                 82565
Pushblock                 386715
Ripper                    305753
Scarifier                 386704
Tip_Control               386715
Tire_Size                 315060
Coupler                   192019
Coupler_System            367724
Grouser_Tracks            367823
Hydraulics_Flow           367823
Track_Type                310505
Undercarriage_Pad_Width   309782
Stick_Length              310437
Thumb                     310366
Pattern_Changer           310437
Grouser_Type              310505
Backhoe_Mounting          331986
Blade_Type                330823
Travel_Controls           330821
Differential_Type         341134
Steering_Controls         341176
saleYear                       0
saleMonth                      0
saleDay                        0
saleDayofWeek                  0
saleDayofYear                  0
dtype: int64
```

Fill missing values

Fill missing numeric values

In [34]:
```
for label, content in df_temp.items():
    if pd.api.types.is_numeric_dtype(content):
        print(label)
```

```
SalesID
SalePrice
MachineID
ModelID
datasource
auctioneerID
YearMade
MachineHoursCurrentMeter
saleYear
saleMonth
saleDay
saleDayofWeek
saleDayofYear
```

In [35]:
```
#check which numeric column have null values
for label, content in df_temp.items():
    if pd.api.types.is_numeric_dtype(content):
        if pd.isnull(content).sum():
            print(label)
```

```
auctioneerID
MachineHoursCurrentMeter
```

In [36]:
```
#fill numeric columns with mean values
for label, content in df_temp.items():
    if pd.api.types.is_numeric_dtype(content):
        if pd.isnull(content).sum():
            #Add binary column which tells us if the data was missing or not
            df_temp[label+"_is_missing"]=pd.isnull(content)
            #fil the missing numeric values with median
            df_temp[label]=content.fillna(content.median())
```

NOTE:

> We use median instead of mean because the median is a more robust measure of central tendency than the mean, which is why it is often used in machine learning.

In [37]:
```
#check again now there is any missing numeric value present or not
for label, content in df_temp.items():
    if pd.api.types.is_numeric_dtype(content):
        if pd.isnull(content).sum():
            print(label)
```

In [38]:
```
#check how many examples are missing before
df_temp.auctioneerID_is_missing.value_counts()
```

Out[38]:
```
auctioneerID_is_missing
False    392562
True      20136
Name: count, dtype: int64
```

Filling and tuning categorical variables into numbers

In [39]:
```
#check aif there is any missing non numeric value present or not
for label, content in df_temp.items():
    if not pd.api.types.is_numeric_dtype(content):
        print(label)
```

```
UsageBand
fiModelDesc
fiBaseModel
fiSecondaryDesc
fiModelSeries
fiModelDescriptor
ProductSize
fiProductClassDesc
state
ProductGroup
ProductGroupDesc
Drive_System
Enclosure
Forks
Pad_Type
Ride_Control
Stick
Transmission
Turbocharged
Blade_Extension
Blade_Width
Enclosure_Type
Engine_Horsepower
Hydraulics
Pushblock
Ripper
Scarifier
Tip_Control
Tire_Size
Coupler
Coupler_System
Grouser_Tracks
Hydraulics_Flow
Track_Type
Undercarriage_Pad_Width
Stick_Length
Thumb
Pattern_Changer
Grouser_Type
Backhoe_Mounting
Blade_Type
Travel_Controls
Differential_Type
Steering_Controls
```

In [40]:
```
pd.Categorical(df_temp["state"]).codes
```

Out[40]:
```
array([43,  8,  8, ...,  4,  4,  4], dtype=int8)
```

You can use the pd.Categorical.codes attribute to get the category codes for any categorical variable in a DataFrame

In [41]:
```
#turn categorical variables into numbers and  filling missing
for label, content in df_temp.items():
    if not pd.api.types.is_numeric_dtype(content):
        #Add a new column (binary) to indicate whether samples has missing values or not
        df_temp[label+"_is_missing"]=pd.isnull(content)
        #Turn categories into numbers and add +1
        df_temp[label]=pd.Categorical(content).codes+1
```

In [42]:
```
pd.Categorical(df_temp["UsageBand"]).codes
```

Out[42]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int8)
```

In [43]:
```
#check aif there is any missing non numeric value present or not
for label, content in df_temp.items():
    if not pd.api.types.is_numeric_dtype(content):
        if pd.isnull(content).sum():
            print(label)
```

In [44]:
```
df_temp.isna().sum()
```

Out[44]:
```
SalesID                       0
SalePrice                     0
MachineID                     0
ModelID                       0
datasource                    0
                             ..
Backhoe_Mounting_is_missing   0
Blade_Type_is_missing         0
Travel_Controls_is_missing    0
Differential_Type_is_missing  0
Steering_Controls_is_missing  0
Length: 103, dtype: int64
```

In [45]:
```
%%time
#Instatiate the model
model= RandomForestRegressor(n_jobs=-1,random_state=42)
#fit the model
model.fit(df_temp.drop("SalePrice",axis=1),df_temp["SalePrice"])
```

```
CPU times: total: 55min 52s
Wall time: 14min 58s
```

Out[45]:
```
RandomForestRegressor(n_jobs=-1, random_state=42)
```

In [46]:
```
#score the model
model.score(df_temp.drop("SalePrice",axis=1),df_temp["SalePrice"])
```

Out[46]:
```
0.9875468079970562
```

Splitting Data into train\valid set

In [47]:
```
#Splitting data
df_val=df_temp[df_temp["saleYear"]==2012]
df_train=df_temp[df_temp["saleYear"]!=2012]
```

In [48]:
```
len(df_val),len(df_train)
```

Out[48]:
```
(11573, 401125)
```

In [49]:
```
#Splitting Data into X and Y
X_train,y_train=df_train.drop("SalePrice",axis=1),df_train.SalePrice
X_valid,y_valid=df_val.drop("SalePrice",axis=1),df_val.SalePrice
```

In [50]:
```
X_train.shape,y_train.shape,X_valid.shape,y_valid.shape
```

Out[50]:
```
((401125, 102), (401125,), (11573, 102), (11573,))
```

In [51]:
```
y_train
```

Out[51]:
```
0          9500.0
1         14000.0
2         50000.0
3         16000.0
4         22000.0
           ...
401120    29000.0
401121    11000.0
401122    11000.0
401123    18000.0
401124    13500.0
Name: SalePrice, Length: 401125, dtype: float64
```

Building an Evaluation Function

In [52]:
```
#Create evaluation function (The competition uses RMSLE)
from sklearn.metrics import mean_squared_log_error, mean_absolute_error,r2_score
def rmsle(y_test,y_preds):
    """
    Calculate root mean squared log error between predictions and true labels
    """
    return np.sqrt(mean_squared_log_error(y_test,y_preds))
#Calculate function to calculate and evaluate model onn different lines
def show_score(model):
    train_preds=model.predict(X_train)
    val_preds=model.predict(X_valid)
    scores={"Training MAE":mean_absolute_error(y_train,train_preds),
            "Valid MAE":mean_absolute_error(y_valid,val_preds),
            "Training RMSLE":rmsle(y_train,train_preds),
            "Valid RMSLE":rmsle(y_valid,val_preds),
            "Training R2 Score":r2_score(y_train,train_preds),
            "Valid R2 Score":r2_score(y_valid,val_preds)}
    return scores
```

Testing our model on a subset (To tune the hyperparameters)

In [53]:
```
#this would take some time....

model=RandomForestRegressor(n_jobs=-1,random_state=42,max_samples=100000)
```

In [54]:
```
%%time
model.fit(X_train,y_train)
```

```
CPU times: total: 19min 3s
Wall time: 5min 7s
```

Out[54]:
```
RandomForestRegressor(max_samples=100000, n_jobs=-1, random_state=42)
```

In [55]:
```
show_score(model)
```

Out[55]:
```
{'Training MAE': 3588.5677039077596,
 'Valid MAE': 6193.733662835912,
 'Training RMSLE': 0.17533085989490504,
 'Valid RMSLE': 0.25674596551158657,
 'Training R2 Score': 0.9389791835237717,
 'Valid R2 Score': 0.8710523572862298}
```

Hyperparameter tuning with RandomizedSearchCV

In [56]:
```
%%time
#Different RandomizedForestRegression Hyperparameter
rf_grid={"n_estimators":np.arange(10,100,10),
         "max_depth":[None,3,5,10],
         "min_samples_leaf":np.arange(1,20,2),
         "min_samples_split":np.arange(2,20,2),
         "max_features":[0.5,1,"sqrt","auto"],
         "max_samples":[10000]}
#Instatiate RandomizedSearchCV model
rs_model=RandomizedSearchCV(RandomForestRegressor(n_jobs=-1,random_state=42),
                            param_distributions=rf_grid,
                            n_iter=2,
                            cv=5,
                            verbose=True)
#Fit the RandomizedSearchCV Model
rs_model.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 2 candidates, totalling 10 fits
CPU times: total: 27.3 s
Wall time: 1min 15s
```

Out[56]:
```
RandomizedSearchCV
estimator: RandomForestRegressor
RandomForestRegressor
```

In [57]:
```
#Finding the best Parameter
rs_model.best_params_
```

Out[57]:
```
{'n_estimators': 90,
 'min_samples_split': 18,
 'min_samples_leaf': 11,
 'max_samples': 10000,
 'max_features': 0.5,
 'max_depth': 5}
```

In [58]:
```
#Evaluate the RandomizedSearchCV model
show_score(rs_model)
```

Out[58]:
```
{'Training MAE': 9800.698169027397,
 'Valid MAE': 11216.714950888701,
 'Training RMSLE': 0.42138318079905046,
 'Valid RMSLE': 0.43719135838940876,
 'Training R2 Score': 0.6239296855704154,
 'Valid R2 Score': 0.6356495217397096}
```

Train a model eith the best hyperparameters

NOTE : These were found after 100 iteration of RandomizedSearchCV

In [59]:
```
%%time
#Most ideal Hyperparameters
ideal_model=RandomForestRegressor(n_estimators=40,
                                  min_samples_split=14,
                                  min_samples_leaf=1,
                                  max_features=0.5,
                                  n_jobs=-1,
                                  max_samples=None,
                                  random_state=42)
#Fit the ideal model
ideal_model.fit(X_train,y_train)
```

```
CPU times: total: 8min 1s
Wall time: 2min 5s
```

Out[59]:
```
RandomForestRegressor
RandomForestRegressor(max_features=0.5, min_samples_split=14, n_estimators=40,
                      n_jobs=-1, random_state=42)
```

In [60]:
```
#show the ideal- model score
show_score(ideal_model)
```

Out[60]:
```
{'Training MAE': 2953.8161137163484,
 'Valid MAE': 5951.247761444453,
 'Training RMSLE': 0.14469006962371858,
 'Valid RMSLE': 0.24524163989538328,
 'Training R2 Score': 0.9588145522577225,
 'Valid R2 Score': 0.8818019502450094}
```

Make Predictions on TEST Data

In [61]:
```
#import test data
df_test=pd.read_csv("bluebook-for-bulldozers/Test.csv",low_memory=False,parse_dates=["saledate"])
df_test.head()
```

Out[61]:

	SalesID	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand	saledate	fiModelDesc	..
0	1227829	1006309	3168	121	3	1999	3688.0	Low	2012-05-03	580G	..
1	1227844	1022817	7271	121	3	1000	28555.0	High	2012-05-10	936	..
2	1227847	1031560	22805	121	3	2004	6038.0	Medium	2012-05-10	EC210BLC	..
3	1227848	56204	1269	121	3	2006	8940.0	High	2012-05-10	330CL	..
4	1227863	1053887	22312	121	3	2005	2286.0	Low	2012-05-10	650K	..

5 rows × 52 columns

Preprocessing the data (Getting the test dataset is the same format as our training set)

In [63]:
```
def preprocess_data(df):
    """
    Perform Transformation in DF and return Transformes DF
    """
    df["saleYear"]=df.saledate.dt.year
    df["saleMonth"]=df.saledate.dt.month
    df["saleDay"]=df.saledate.dt.day
    df["saleDayofWeek"]=df.saledate.dt.dayofweek
    df["saleDayofYear"]=df.saledate.dt.dayofyear

    df.drop("saledate",axis=1,inplace=True)

    #Fill numeric rows with median
    for label, content in df.items():
        if pd.api.types.is_numeric_dtype(content):
            if pd.isnull(content).sum():
              #Add binary column which tells us if the data was missing or not
              df[label+"_is_missing"]=pd.isnull(content)
              #fil the missing numeric values with median
              df[label]=content.fillna(content.median())

        #fill categorical missing data and turned categorics into numbers
        if not pd.api.types.is_numeric_dtype(content):
              #Add a new column (binary) to indicate whether samples has missing values or not
              df[label+"_is_missing"]=pd.isnull(content)
              #Turn categories into numbers and add +1
              df[label]=pd.Categorical(content).codes+1

    return df
```

In [64]:
```
#process the test dat
df_test=preprocess_data(df_test)
```

In [65]:
```
df_test.head()
```

Out[65]:

	SalesID	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand	fiModelDesc	fiBaseMode
0	1227829	1006309	3168	121	3	1999	3688.0	2	499	18
1	1227844	1022817	7271	121	3	1000	28555.0	1	831	29
2	1227847	1031560	22805	121	3	2004	6038.0	3	1177	40
3	1227848	56204	1269	121	3	2006	8940.0	1	287	11
4	1227863	1053887	22312	121	3	2005	2286.0	2	566	19

5 rows × 101 columns

In [67]:
```
df_test["auctioneerID_is_missing"]=False
```

In [69]:
```
df_test.head()
```

Out[69]:

	SalesID	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand	fiModelDesc	fiBaseMode
0	1227829	1006309	3168	121	3	1999	3688.0	2	499	18
1	1227844	1022817	7271	121	3	1000	28555.0	1	831	29
2	1227847	1031560	22805	121	3	2004	6038.0	3	1177	40
3	1227848	56204	1269	121	3	2006	8940.0	1	287	11
4	1227863	1053887	22312	121	3	2005	2286.0	2	566	19

5 rows × 102 columns

In [81]:
```
# IF NECESSARY
#model.fit(X_train,y_train)
```

Out[81]:

RandomForestRegressor

```
RandomForestRegressor(max_samples=100000, n_jobs=-1, random_state=42)
```

In [83]:
```
df_test=df_test[model.feature_names_in_]
```

In [ ]:
```
#MAke prediction on test data
test_preds=ideal_model.predict(df_test)
```

In [85]:
```
test_preds
```

Out[85]:
```
array([17030.00927386, 14355.53565165, 46623.08774286, ...,
       11964.85073347, 16496.71079281, 27119.99044029])
```

We've made some predictions but they're not in the same format Kaggle is asking for :

> > https://www.kaggle.com/c/bluebook-for-bulldozers/overview/evaluation

In [86]:
```
# Format predictions into the same format Kaggle is after
df_preds = pd.DataFrame()
df_preds["SalesID"] = df_test["SalesID"]
df_preds["SalesPrice"] = test_preds
df_preds
```

Out[86]:

	SalesID	SalesPrice
0	1227829	17030.009274
1	1227844	14355.535652
2	1227847	46623.087743
3	1227848	71680.261335
4	1227863	61762.999424
...	...	...
12452	6643171	39966.363007
12453	6643173	12049.704433
12454	6643184	11964.850733
12455	6643186	16496.710793
12456	6643196	27119.990440

12457 rows × 2 columns

In [88]:
```
# Export prediction data
df_preds.to_csv("bluebook-for-bulldozers/test_predictions.csv", index=False)
```

Feature Importance

Feature Importance seeks to figure out whuch different attributes of the data were most importance when it comes to predicting the target variables(SalePrice)

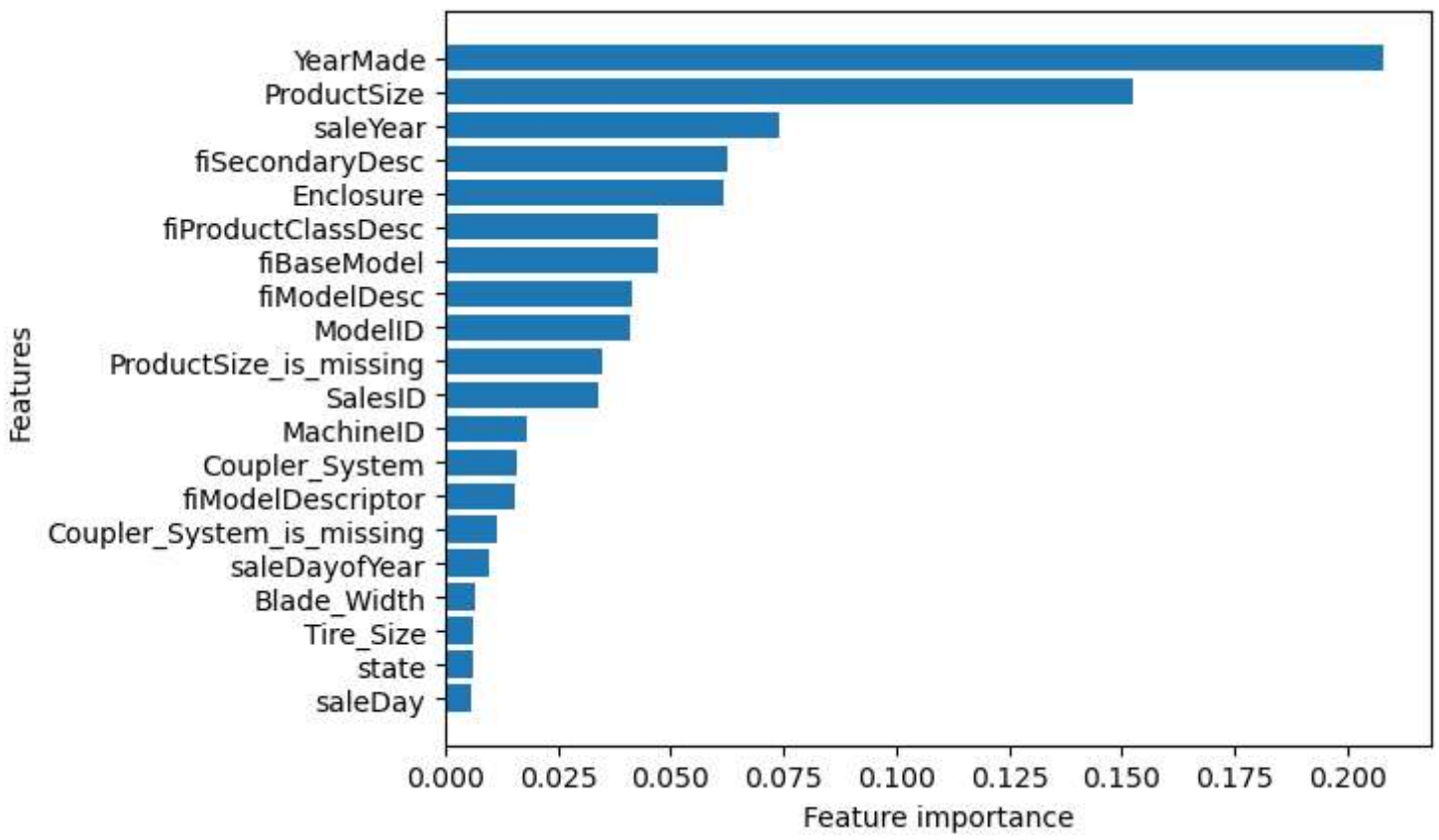
In [90]:
```
#find features importance of our best model
ideal_model.feature_importances_
```

Out[90]:
```
array([3.39445533e-02, 1.81148281e-02, 4.09167072e-02, 1.70752171e-03,
       3.40797459e-03, 2.08200698e-01, 2.95067052e-03, 1.10113725e-03,
       4.16122668e-02, 4.71911805e-02, 6.23815431e-02, 4.67433955e-03,
       1.52524442e-02, 1.52517337e-01, 4.72224713e-02, 5.96817956e-03,
       1.29351899e-03, 2.78088439e-03, 2.37248769e-03, 6.17114453e-02,
       8.13525488e-04, 3.61873268e-05, 9.19098115e-04, 2.23170993e-04,
       1.28102678e-03, 2.06519636e-05, 2.01477316e-03, 6.63364759e-03,
       2.15274492e-03, 2.50178165e-03, 4.63902393e-03, 3.85873985e-03,
       2.76062667e-03, 1.00782454e-03, 2.47969268e-04, 6.04239818e-03,
       7.64997072e-04, 1.57100537e-02, 2.29716203e-03, 2.58372272e-03,
       8.07637426e-04, 9.18548690e-04, 1.35656446e-03, 5.81458569e-04,
       4.96716928e-04, 3.79552257e-04, 5.31712788e-04, 2.71823509e-03,
       8.34294376e-04, 3.12136841e-04, 2.14075157e-04, 7.42422919e-02,
       3.80158492e-03, 5.67641024e-03, 2.87154703e-03, 9.83349904e-03,
       2.65470837e-04, 1.57946459e-03, 3.10058108e-04, 0.00000000e+00,
       0.00000000e+00, 2.27421721e-03, 1.05632062e-03, 5.42819222e-03,
       3.48484864e-02, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 1.90858845e-05, 9.09490682e-06, 1.31265147e-04,
       5.29163902e-06, 1.11952381e-04, 4.78452431e-06, 3.43582863e-04,
       5.57068428e-06, 1.07167376e-03, 3.99179008e-03, 4.07753410e-03,
       1.05749617e-04, 2.76528927e-03, 2.59244312e-05, 3.51888176e-04,
       2.31519337e-03, 1.99211177e-03, 4.02034629e-03, 2.03778082e-04,
       1.13483313e-02, 9.02551628e-04, 1.58182497e-03, 4.63243398e-05,
       2.92071004e-04, 3.11923094e-05, 1.56873538e-04, 2.87205987e-05,
       3.80543083e-05, 2.55045807e-04, 1.66878572e-04, 2.10341792e-04,
       1.26024842e-04, 9.40663015e-05])
```

In [98]:
```
# Helper function for plotting feature importance
def plot_features(columns, importances, n=20):
    df = (pd.DataFrame({"features": columns,
                        "feature_importances": importances})
          .sort_values("feature_importances", ascending=False)
          .reset_index(drop=True))

    # Plot the dataframe
    fig, ax = plt.subplots()
    ax.barh(df["features"][:n], df["feature_importances"][:20])
    ax.set_ylabel("Features")
    ax.set_xlabel("Feature importance")
    ax.invert_yaxis()
```

In [100…
```
plot_features(X_train.columns, ideal_model.feature_importances_)
```



In [101…
```
df["Enclosure"].value_counts()
```

Out[101…
```
Enclosure
OROPS                  177971
EROPS                  141769
EROPS w AC              92601
EROPS AC                   18
NO ROPS                     3
None or Unspecified         2
Name: count, dtype: int64
```

__Question to finish__: Why might knowing the feature importances of a trained machine learning model be helpful?

__Final challenge/extension__: What other machine learning models could you try on our dataset?

> Hint: https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

check out the regression section of this map, or try to look at something like CatBoost.ai or XGBooost.ai.

In [ ]: